1985 год - создание IA - 32. От неё пошли:

1. IA - 64 - полностью изменили все команды, не было обратной совместимости
2. AMD64 - расширение для IA-32 (вышла в 2000)

Термины:
FLOPS - float operation in second
АЛУ - устройство для работы с целыми числами
FPU - устройство для работы с дробными числами

Дилема создания приложения:
Если мы пишем приложение, то есть два пути:

1. Создать приложение для всех (используя только AMD64). Оно будет несовременное, но для каждого
2. Создать современное приложения (используя наборы команд помимо AMD64), но оно будет не для всех

Решение 1:
Автоматическая диспетчеризация

Приложение компилируется два раза, а потом auto dispatch (компонент приложения) спрашивает у процессора, что он может и, в зависимости от ответа, запускает нужный проект.

Минусы:

1. В n раз больше кода требуется писать
2. Приложение собирается в n раз дольше
3. Занимает много кэша
4. Весит приложение больше

Оптимизация

Вычисление точности чисел с плавающей запятой
Строгие вычисления с плавающей запятой по стандартом ANSI C или IEEE.
В процесоре есть ядро для работы с числами с плавающей запятой (FPU). Он принимает X, Y и выдаёт число Z (максимальной точности).
Если точность не очень нужна, то все операции деления, можно заменить на

обратное умнажение, что позволяет ускорить работу приложения
Точности и их вес в байтах:

1. single - 4
2. double - 8
3. extended - 10
4. quadrable - 10

Опции оптимизации должны выбираться для каждого исходного фвйла индивидуально!
Файл сборки не должны зависить от среды разработки!

Link Time Optimization (LTO) - оптимизация, когда компилируется, например, не вся библиотека, а только некоторые функции, чтобы приложение работали быстрее.

-Qprec_div - оптимизация, которая деление превращает в умножение на обратное число, что уменьшает точность, но ускоряет работу
-Qlong_double - оптимизация, которая уменьшает точность вычислений, но ускоряет работу приложения

Системные методы оптимизации приложений

Оптимизация высокого уровня

```cpp
for (j = 1; j <1000; j++){
        y(j) = y(j) + a*x(j);
}
```

использует ldf (load floating-point)

```cpp
for (j = 1; j < 1000; j+=2){
        y(j) = y(j) + a * x(j);
        y(j+1) = y(j+1) + a * x(j+1);
}
```

ldfp (load floating-point pair)

Второй цикл более производителен

Подробнее об оптимизации высокого уровня:

- High Level Optimizier (HLO)
- HLO ориентирован на максимальную скорость работы

- Может переписать алгоритм, чтобы максимально увеличить количество успешных обращений к кэш-памяти, осуществляя более агресивный анализ зависимости по данным
- Пример:
  gcc -O3 superprog.cpp

**Ограничения оптимизации выского уровня:

- Циклы должны удовлетворять тем же требования, что и для векторизации
- HLO необязательно повысит производительность некоторых приложений

Как работает IPO?

- Interprocedural Optimization (IPO)
- Функции оптимизируется внутри файла

```language-cpp
int func(int x, int, y, int z){
        operator
}

int main(){
        for (int i = 0; i < N; i++){
                ...
                func(x, y, z);
                ...
        }
}
```

Компилятор сам определять прописывать ли модификатор inline к функции

Оптиции:

- -ip (между процедурами)
- -ipo (между процедурами и встраивами файлами)

Ограничения:

- IPO может увиличить размер кода, что может заставить процессор использовать кэш менее эфективно
- Компиляция может занимать намного больше времени

Ведомая профилем оптимизация

- Profile Guided Optimization (PGO)

- Такую компиляцию можно назвать:
  - динамической
  - замкнутой
  - или компиляцией с обратной связью
- Предоставлять компилятору информацию о динамическом управлении программой для выполнения наилучший вариантов оптимизаций
- Используются различные статические и вероятностные методы

Этапы:

1. Instrumented compilation
2. Instrumented execution
3. Feedback compilation and linking

Опции:

- Компилятор gcc
  - -fprofile-generate
  - -fprofile-use
- Компиляторы Intel
  - -prof_gen[x]
  - -prof_use
  - Пример

```
gcc -fprofile-generate test.c -o test
test
gcc -fprofile-use test.c -o test
```

Пример многократного прифилирования

1. *Инструментальная компиляция*
   ```
   icl -prof_gen -e pgo.exe main.c other.c
   ```
2. *Инструментальное выполнения*
   ```
   pgo.exe input1
   pgo.exe input2
   pgo.exe input3
   ```
3. **Компиляция с обратной связью**
   ```
   icl -prof_use -e pgo main.c other.c
   ```

Основы векторизации кода

Приобретаемая компитенция

- Применение знаний и умений по написанию кода на *языке высокого уровня* для *современных архитектур процессоров*
- Применяется в технологиях программирования и без регулярного программирования
- Регулярное программирование позволяет *легче задействовать* веторизации

Одни из показателей кода

- Производительность кода **Perf** (code performance) - количественный показатель
  - Производительность участка кода: **Perf = 1 / T**
- Потребляемая память Mem (memory consumption) - количественный показатель
- Поддержка современных микроархитектур - *качественный* показатель
- Все три относятся к пользователю
- Сложность кода Compl (code complexity) - количественный показатель
  Compl = LOC
  Compl = Cycl
- Число строк кода (LOC)
- Цикломантическая сложность кода Cycl (cyclomatic complexity of a code) - количество линейно независимый *маршрутов* через программный код
- Относится к **разработчику**

Эффективность кода

- Интегральный показатель
- Относится к пользователю разработчику
  - Эффективность кода **Prod** (code productivity): *Prod = Perf / Compl = 1 / (T* Compl)*
- Если код в два раза больше при сравнимой производительности, то он в два раза *менее эффективнее*
- Если код в два раза *более производителен* и в два раза больше, то его эффективность **не улучшена**

Высоко-эффективные технологии

- High-productivity computing

- Технологии, направленные на повышение эффективности кода, а не только производительности

Гетерогенная архитектура - архитектура, у которой разнородные вычислители. Например, GPGPU, FPGA

Эффективность процессора

- Производительность Perf (performance)
- Рассеиваемая *мощность* TDP (Therma; design power)
- Эффективность Prod (productivity): **Prod = Perf / TDP**
- В знаменатель можно добавить:
  - стоимость
  - оказание влияние на окружающую среду при разработке и эксплуатации